WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Design of a custom frequency divider for aviation equipment using the Kit FPGA Altera DE2

Phan Hien[1], Vuong Thuy Linh [2] , Le Ngoc Giang[3*]

[1]Master, Head of the Radar Basics Department, Faculty of Fundamental Technics, AD-AF Academy of Viet Nam, Ha Noi, Vietnam
[2]Master, Lecturer, Department of Information Technology, Faculty of General Education, University of Labour and Social Affairs, No. 43, Tran Duy Hung Street, Trung Hoa Ward, Cau Giay District, Hanoi City, Vietnam
[3]Doctor, Head of the Metrology Department, Faculty of Fundamental Technics, AD-AF Academy of Viet Nam, Ha Noi, Vietnam
** Email: lengocgianglinh@gmail.com*
DOI: https://doi.org/10.55248/gengpi.2022.3.4.15

A B S T R A C T

When operating aviation equipment or when working on test benches, test tables, etc., it is necessary to choose the right frequency to use. The custom frequency divider was designed by the author with the purpose of giving the right frequency to the desired frequency. A custom frequency divider was installed on Kit Altera DE2. The frequency selection is set by the user via push buttons.

Keywords: Kit FPGA Altera DE2, custom frequency divider, aviation equipment, Quartus II.

1. Introduction

Due to the simple refactoring ability and possessing a large block of logic resources, FPGA is applied to many classes of problems, such as: digital signal processing and digital filtering; signal modulation and demodulation; encoding, decoding, and speech recognition; audio signal processing and digital image processing; applications in information systems such as voice IP systems, voice mail, modems, mobile phones, communication in LAN, WIFI, television, and radio; applications in controlling electronic devices: hard drives, printers, industrial machines, navigation, positioning, and robots.

Using the DE 2 Kit is especially important in the design of electronic circuits such as frequency dividers and failure detectors used on aircraft. Design a failure warning and a pilot instruction circuit on the aircraft: when a failure signal is sent to the warning unit, it will perform a warning process by sound, image, and failure indicator light on the aircraft. When building a program, we can use the DE2 Kit to check the working process of the system with the hardware integrated into the Kit. Design a frequency divider circuit: Create a frequency divider circuit to generate corresponding frequencies in aeronautical engineering to meet test requirements and frequency value requirements.

In addition, modern aircraft have used FPGA technology and applied Altera microchips in automatic control systems, remote control systems, inertial guidance systems, and weapons control systems to solve the problems of aiming and guiding, optimal selection of aeronautical lethal vehicles, and image processing to display on the screen to notify the pilot. Altera's FPGA components such as FLEX8000, FLEX10K, CYCLONE, and STRATIX are used very effectively in RAM and ROM program memories in computers. Altera components are also used a lot in ground test vehicles, missile test equipment design, etc., and have demonstrated technological superiority, such as: integration ability and working accuracy, the ability to check and replace when there is a breakdown.

* *Corresponding author: Le Ngoc Giang*. Tel.: +84969896136
E-mail address: lengocgianglinh@gmail.com

2. Design of a custom frequency divider

2.1. Structure of a custom frequency divider

The self-tuning frequency divider uses the DE2 Kit, which is designed for the user to set the desired output frequency by pressing the 1÷4 Buttons. After pushing the Start switch to start dividing the frequency, the output of the Clk-Out pin will have the desired frequency in the range of 1÷9999Hz. The self-tuning frequency divider is designed with modules as shown in Figure 1.

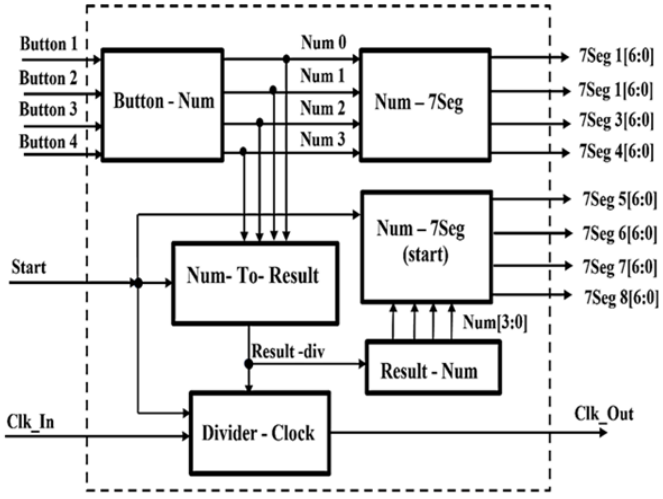

Fig. 1- Structure diagram of a custom frequency divider

-Button-Num: The module identifies the button, converts it to the corresponding integer value.

-Num-7seg: The module displays the output frequency value of the divider, converting it into 7-bit binary code for display on a 7-segment LED.

- Num-to-Result: The module calculates the output frequency value from the input number push button value.

- Divider-Clock: Frequency division module, with 3 inputs: Clk_in with a frequency of 50MHz, Result_div with a frequency value to be divided, Start; Output is Clk_out.

- Result-Num: The module calculates and displays the set frequency value.

2.2. The general principle of custom frequency dividers

A frequency divider is an important component in digital circuit design and is used very commonly. The frequency divider is used to generate the desired frequency by dividing the clock input pulse frequency by a certain factor. The frequency divider has the frequency of the input signal (fin) and the frequency of the output signal (fout). They are related by the scale factor N:

$$f_{out} = f_{in} / N; \quad \text{or} \quad T_{out} = T_{in} * N$$

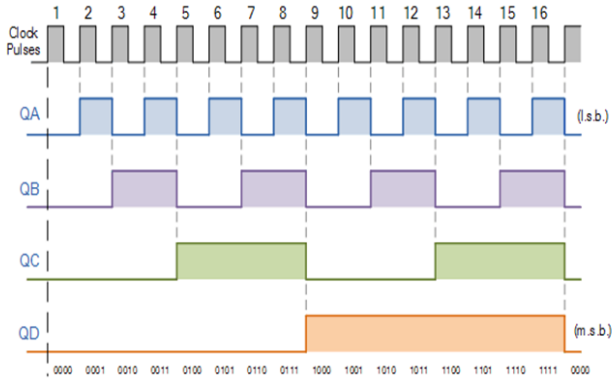

Fig.2 - Custom frequency divider pulse plot

2.3. Custom Frequency Divider Main Program

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
ENTITY Divider IS
	port(
		KEY: in std_LOGIC_vector(3 downto 0);
		Clk_in: in std_logic;		- - CLOCK_50
		Start : in  STD_LOGIC;		- - SW0
		HEX0: out std_logic_vector(6 downto 0);
		HEX1: out std_logic_vector(6 downto 0);
		HEX2: out std_logic_vector(6 downto 0);
		HEX3: out std_logic_vector(6 downto 0);
		Clk_out: out std_logic;
		Clk_out2: out std_LOGIC  );
END Divider;
ARCHITECTURE Structure OF Divider IS
	signal Num0: integer:=0;
	signal Num1: integer:=0;
	signal Num2: integer:=0;
	signal Num3: integer:=0;
	signal Result_div: integer:=0;
component button_num
	port(	button: in std_logic;
		clk:	in std_logic;
		number: out integer);
end component;
component num_seg_on
	port(number: in integer;
		clk:	in std_logic;
		SEGMENT: out std_logic_vector(6 downto 0));
end component;
component num_seg_start
	port(number: in integer;
		clk:	in std_logic;
		button_start: in std_logic;
		SEGMENT: out std_logic_vector(6 downto 0));
END component;
component button_result_div
	Port (	num0,num1,num2,num3: in integer;
		 clk:	in std_LOGIC;
		start:  in std_LOGIC;
		result_div: out integer );
end component;
component divider_clock
	Port (	clk_in : in  STD_LOGIC;
		start:	in std_LOGIC;
		 result_div: in integer;
		clk_out:  out STD_LOGIC;
		clk_out2: out std_LOGIC		);
end component;
BEGIN
--  -- Calculating of the number
Number0: button_num port map (button => KEY(0),clk => Clk_in, number => Num0);
Number1: button_num port map (button => KEY(1),clk => Clk_in, number => Num1);
Number2: button_num port map (button => KEY(2),clk => Clk_in, number => Num2);
```

```
Number3: button_num port map (button => KEY(3),clk => Clk_in, number => Num3);
--  -- Display the value of the frequency to divide
Display00: num_seg_on port map (number => Num0, clk=>Clk_in, SEGMENT => HEX0);
Display01: num_seg_on port map (number => Num1, clk=>Clk_in, SEGMENT => HEX1);
Display02: num_seg_on port map (number => Num2, clk=>Clk_in, SEGMENT => HEX2);
Display03: num_seg_on port map (number => Num3, clk=>Clk_in, SEGMENT => HEX3);
--  -- Calculating the value of the frequency to divide
Result_Divider: button_result_div port map (num3 => Num3, num2 => Num2, num1 => Num1, num0 => Num0, clk => Clk_in,start => Start, result_div => Result_div);
--  -- Calculate the output frequency value
DIVIDER: divider_clock port map (clk_in => Clk_in,start => Start, result_div => Result_div, clk_out => Clk_out,clk_out2 => Clk_out2);
        END Structure;
```

3. Divider-Clock frequency division module

3.1. Design of a Custom Frequency Division Module

Divider-Clock: Frequency division module, with 3 inputs: Clk_in with a frequency of 50MHz, Result_div with a frequency value to be divided, Start; Output is Clk_out. A custom frequency divider, designed to work only when started with the Start button. The output frequency has a pulse width of 50%. The value (2*N) is the frequency division factor. The output frequency is calculated as follows:

$$FOUT = FIN /(2*N);$$

The flowchart of the custom frequency divider is shown in Figure 3.

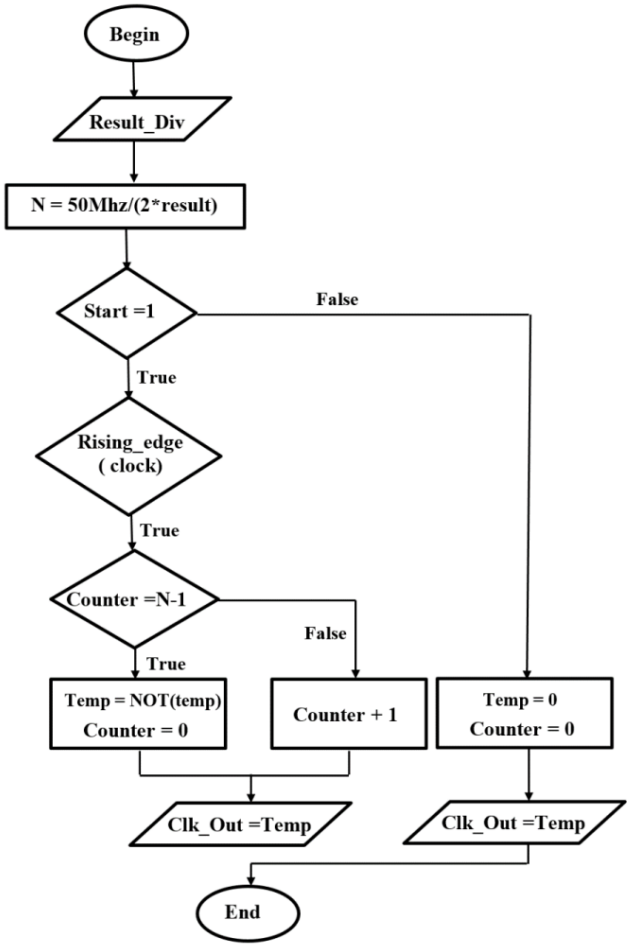

Fig.3- The flowchart of the custom frequency divider

3.2. Program code of the custom frequency division module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity divider_clock is
        Port (
                clk_in : in  STD_LOGIC;
                 start:   in std_LOGIC;
                 result_div: in integer;
                clk_out:  out STD_LOGIC;
                 clk_out2: out std_LOGIC
        );
end divider_clock;
architecture Behavioral of divider_clock is
         signal temporal: STD_LOGIC;
         signal counter : integer;
         signal value_div:integer;
begin
        value_div <= 50000000/(2*result_div);
        frequency_divider: process (start, clk_in)
         begin
                        if(start = '1') then
                                if rising_edge(clk_in) then
                                        if (counter = (value_div-1)) then
                                                temporal <= NOT(temporal);
                                                counter  <= 0;
                                        else
                                                counter <= counter + 1;
                                        end if;
                                end if;
                        else
                                temporal <= '0';
                                counter  <= 0;
                        end if;
                                 clk_out  <= temporal;
                                 clk_out2 <= temporal;
        end process;
end Behavioral;
```

4. Button-Num button recognition module

4.1. Flowchart of the button recognition program algorithm

The push button recognition module is responsible for recognizing the status of the buttons on the DE2 Kit (when the button is pressed, the corresponding logic is 0). Convert the corresponding button's value to the integer value. The corresponding value of each button is 0÷9 and will be displayed on the 7-segment LED, so we only need to recognize the state of the push buttons. The flowchart of the program to recognize the push buttons is converted to corresponding integer values 0÷9, as shown in Figure 4.

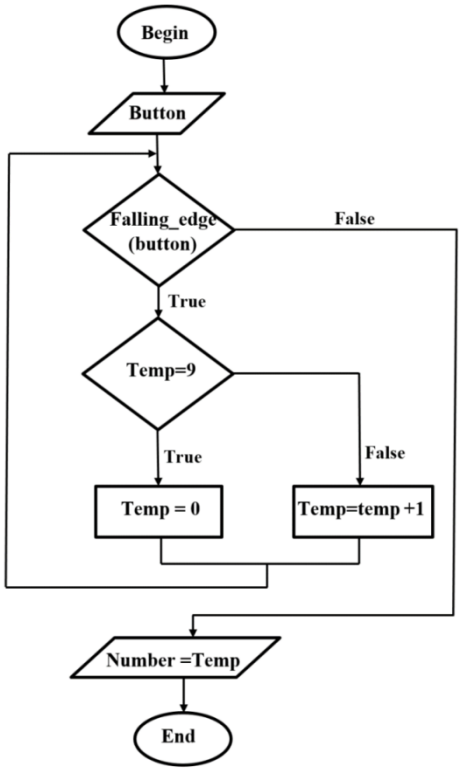

Fig.4- Flowchart of the button recognition program algorithm

4.2. Program code of the button recognition module

```
        LIBRARY IEEE;
        USE IEEE.STD_LOGIC_1164.ALL;
        USE IEEE.STD_LOGIC_UNSIGNED.ALL;
        USE IEEE.STD_LOGIC_ARITH.ALL;
ENTITY button_num IS
         port (
button: in std_logic;
                clk:    in std_logic;
                number: out integer
                     );
END button_num;
ARCHITECTURE Behavioral OF button_num IS
        signal temp: integer range 0 to 9 := 0;
BEGIN
        process(clk)
        BEGIN
                        if(falling_edge(button))then
                                        if(temp = 9) then
                                                temp <=0;
                                        else
                                                temp <= temp+1;
                                        end if;
                        end if;
                number <= temp;
        END process;
END Behavioral;
```

5. Num-to-Result Frequency Value Calculation Module

5.1. The Calculation of Frequency Value Principle

The frequency value calculation module has the task of calculating the output frequency value to be divided from the value of the buttons. With four buttons, there will be four corresponding values: Num3, Num2, Num1, and Num0. The principle of calculating the output frequency value according to the formula:

Result_div = (1000*Num3) + (100*Num2) + (10*Num1) + Num0.

5.2. The frequency value calculation module's program code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity button_result_div is
                Port (      num0,num1,num2,num3: in integer;
                            clk:   in std_LOGIC;
                            start:  in std_LOGIC;
                            result_div: out integer);
end button_result_div;
architecture Behavioral of button_result_div is                                  BEGIN
                                                                                 process (clk,start)
                begin
                        if rising_edge(start) then
result_div <= (1000*num3)+(100*num2)+(10*num1)+num0;
                        end if;
        end process;
END Behavioral;
```

6. The module displays the frequency value on a 7-segment LED: Num-7seg

6.1. Operating principle

Module Num-7seg works on the principle of converting each number value into a 7-bit binary value for display on a 7-segment LED. On the DE2 Kit, the 7-segment LED is connected to a common anode. When the LED input pin has bit 1, the LED is off. When the LED input pin has bit 0, the LED lights up.

For example, if the value of Number = 0, then the 7-bit code on the 7-segment LED is: "1000000"; if Number = 1, then the 7-bit code on the 7-segment LED is: "1111001".

6.2. Program code of the 7-segment LED display module

```
LIBRARY IEEE;
        USE IEEE.STD_LOGIC_1164.ALL;
        USE IEEE.STD_LOGIC_UNSIGNED.ALL;
        USE IEEE.STD_LOGIC_ARITH.ALL;
ENTITY num_seg_on IS
                 port(
                        number: in integer;
                        clk:   in std_logic;
                        SEGMENT: out std_logic_vector(6 downto 0)
                        );
END num_seg_on;
ARCHITECTURE Behavioral OF num_seg_on IS
BEGIN
                process(clk,number)
                BEGIN
                        if (clk='1') then
```

```
case number is
when 0 => SEGMENT <= "1000000"; -- 0
when 1 => SEGMENT <= "1111001"; -- 1
when 2 => SEGMENT <= "0100100"; -- 2
when 3 => SEGMENT <= "0110000"; -- 3
when 4 => SEGMENT <= "0011001"; -- 4
when 5 => SEGMENT <= "0010010"; -- 5
when 6 => SEGMENT <= "0000010"; -- 6
when 7 => SEGMENT <= "1111000"; -- 7
when 8 => SEGMENT <= "0000000"; -- 8
when 9 => SEGMENT <= "0010000"; -- 9
when others => SEGMENT <= "1111111";
                                end case;
                        end if;
        END process;
END Behavioral;
```

7. Test and evaluate the custom frequency divider's quality

After programming, compiling, and simulating the custom frequency divider on Quartus II software, the program is loaded and tested experimentally with the DE2 Kit.

Here are the simulation and experimental results:

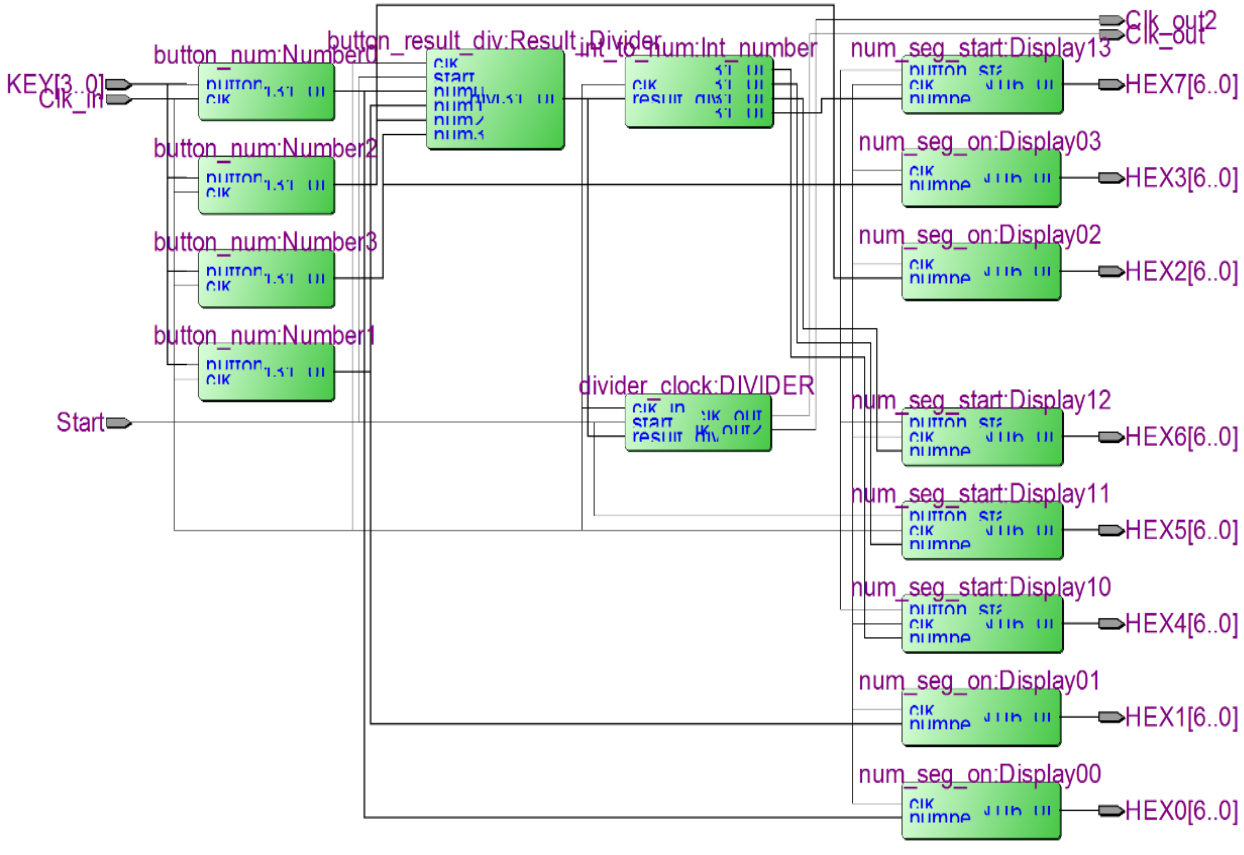

Fig.5- RTL register transfer level design on Quartus 2

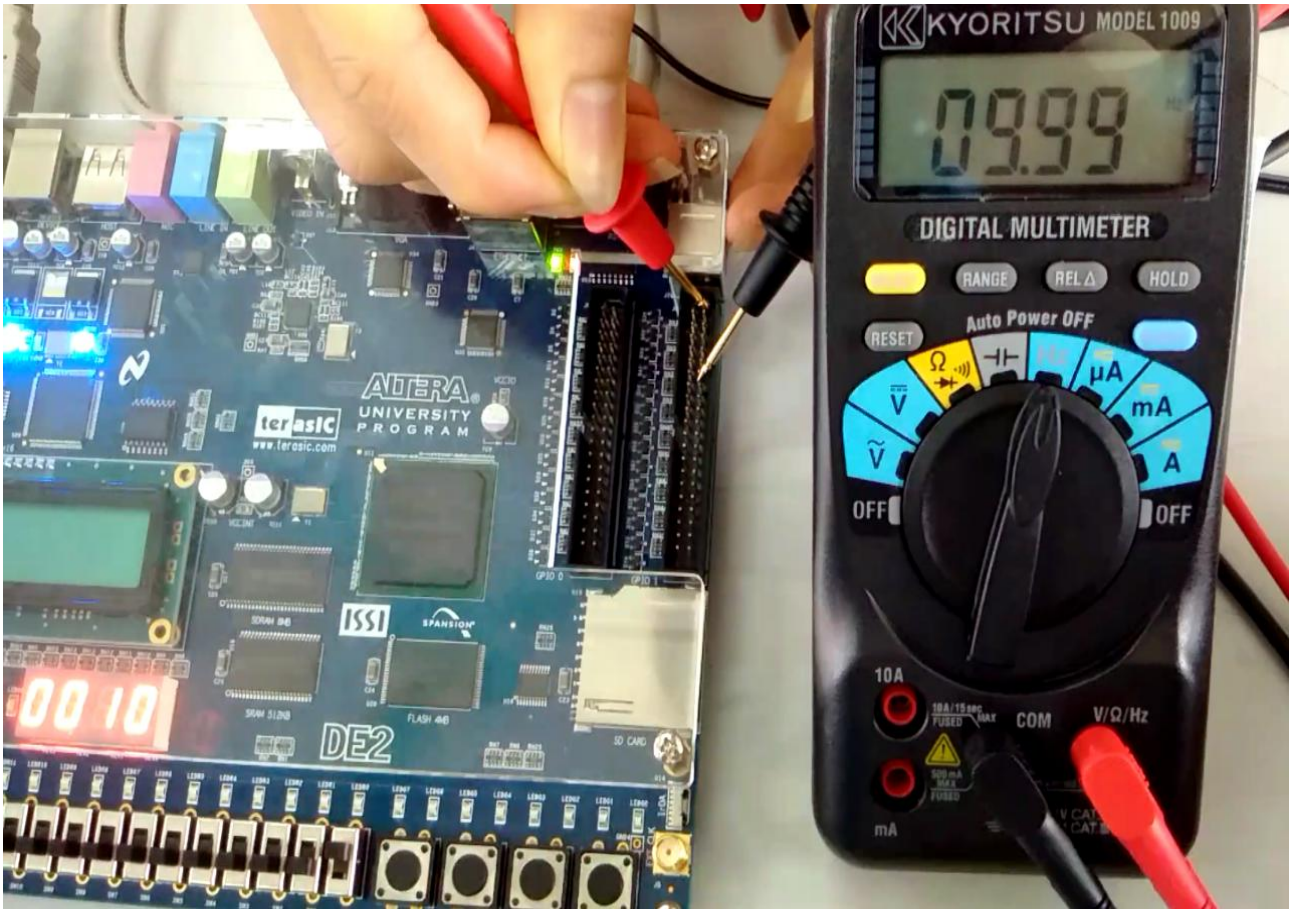

Fig.6- Experimental test on the DE2 kit

Check the output frequency through 2 experiments:

- Observe the red LED status on the DE2 Kit;
- Measure the output frequency.

The test results show that the custom frequency divider meets the original design requirements. The output frequency is exactly the same as the frequency set by the user via the push button.

4. Conclusion

In the article, the authors clarify the principle of building a frequency divider from the input frequency, programming, compiling, and loading it on the FPGA Altera DE2 Kit. The authors designed a custom frequency divider that produces output frequency values in the range of 1–99,999 Hz. Simulation results and experimental tests show that the custom frequency divider works well; the output frequency measured on the Clk-Out pin is correct with the value set on the buttons. If the number of buttons is increased, a custom frequency divider can be set up with a larger frequency setting value.

ACKNOWLEDGEMENTS

This work is supported by: Department of Information Technology, Faculty of General Education, University of Labour and Social Affairs in Hanoi Vietnam. Faculty of Fundamental Technics, AD-AF Academy of Viet Nam.

REFERENCES

Roger Lipsett, Carl F. Schaefer, Cary Ussery (1989). VHDL: Hardware Description and Design. *Kluwer Academic Publishers, United States of America*.

Wayne Wolf (2004). FPGA-Based System Design. *Prentice Hall.*

Harold Thimble by (2013). Reasons to Question Seven Segment Displays. *CHI 2013.*

Shengwei Meng, Zhenghuan Xia (2014). An FPGA-Integrated Time-to-Digital Converter Based on a Ring Oscillator for Programmable Delay Line Resolution Measurement. *Journal of Electrical and Computer Engineering*.

K. J. Hong, E. Kim, J. Y. Yeom (2012). FPGA-based time-to-digital converter for time-of-flight PET detector. *In Proceedings of the Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC '12)*, pp. 2463–2465.

S. S. Junnarkar, P. O'Connor, P. Vaska, (2009). FPGA-based self-calibrating time-to-digital converter for time-of-flight experiments. *IEEE Transactions on Nuclear Science*, vol. 56, no. 4, pp. 2374–2379.

P. Chen, P.-Y. Chen, J.-S. Lai (2010). FPGA vernier digital-to-time converter with 1.58 ps resolution and 59.3 minutes operation range. *IEEE Transactions on Circuits and Systems*, vol. 57, no. 6, pp. 1134–1142.